NP Completeness

Cost and Complexity

- **Algorithm complexity can be expressed in Order notation, e.g. "at what rate does work grow with N?":**
  - **O(1) Constant**
  - **O(logN) Sub-linear**
  - **O(N) Linear**
  - **O(NlogN) Nearly linear**
  - **O(N^2) Quadratic**
  - **O(X^N) Exponential**

- **But, for a given problem, how do we know if a better algorithm is possible?**

The Problem of Sorting

For example, in discussing the problem of sorting:

- **Two algorithms to solve:**
  - **Bubblesort – $O(N^2)$**
  - **Mergesort – O(N Log N)**
- **Can we do better than O(N Log N)?**

Algorithm vs. Problem Complexity

- **Algorithmic complexity is defined by analysis of an algorithm**

- **Problem complexity is defined by**
  - **An upper bound – defined by an algorithm**
  - **A lower bound – defined by a proof**

The Upper Bound

- **Defined by an algorithm**
- **Defines that we know we can do at least this good**
- **Perhaps we can do better**
- **Lowered by a better algorithm**
  - **"For problem X, the best algorithm was $O(N^3)$, but my new algorithm is $O(N^2)$."**

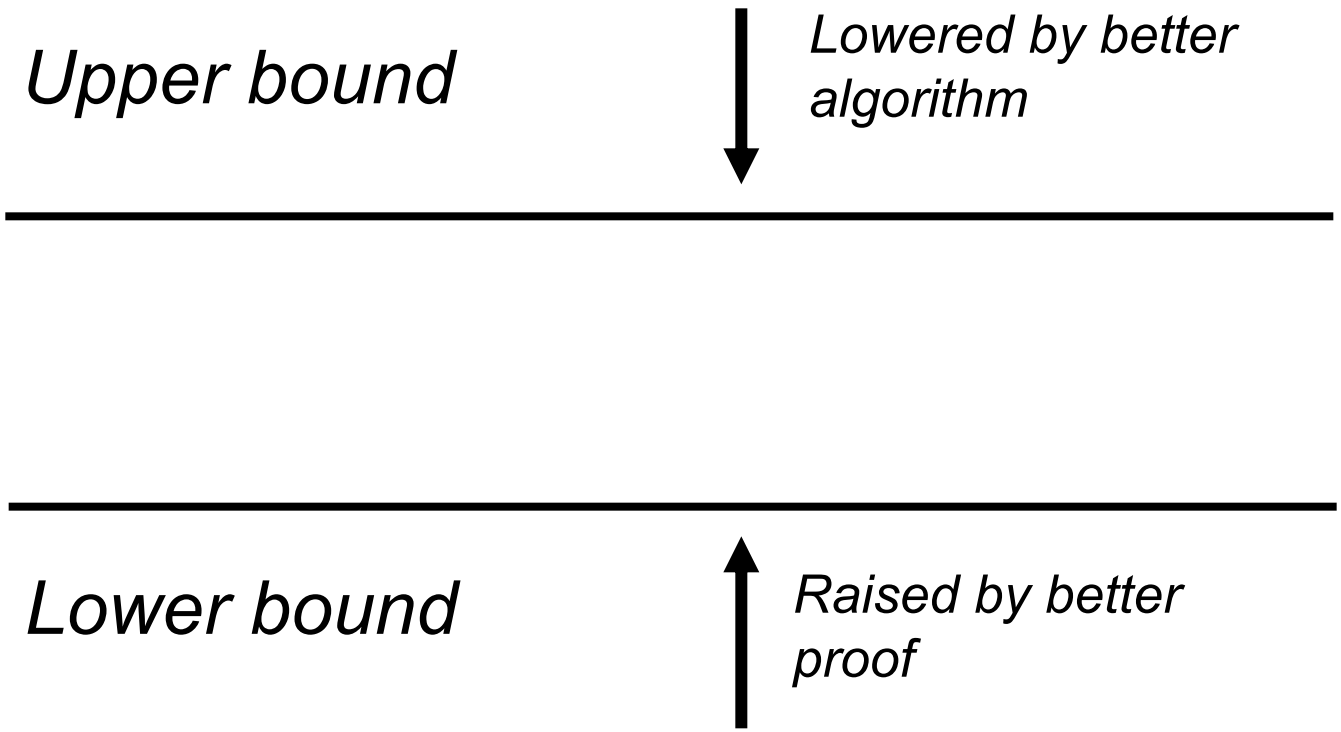
The Lower Bound

- **Defined by a proof**
- **Defines that we know we can do no better than this**
- **It may be worse**
- **Raised by a better proof**
  - **"For problem X, the strongest proof showed that it required O(N), but my new, stronger proof shows that it requires at least $O(N^2)$."**

Upper and Lower Bounds

- **The Upper bound is the best algorithmic solution that has been found for a problem.**
  - **"What's the best that we know we can do?"**
- **The Lower bound is the best solution that is theoretically possible.**
  - **"What cost can we prove is necessary?"**

Changing the Bounds



Open Problems

The upper and lower bounds differ.

Upper bound

Lowered by better algorithm

Unknown

Lower bound

Raised by better proof

Closed Problems

The upper and lower bounds are identical.

Upper bound

Lower bound

Closed Problems

- **Better algorithms are still possible**
- **Better algorithms will not provide an improvement detectable by "Big O"**
- **Better algorithms can improve the constant costs hidden in "Big O" characterizations**

Tractable vs. Intractable

Problems are tractable if the upper and lower bounds have only polynomial factors.

- **O (log N)**
- **O (N)**
- **O (N^K) where K is a constant**

Problems are intractable if the upper and lower bounds have an exponential factor.

- **O (N!)**
- **O (N^N)**
- **O (2^N)**

Terminology

- **Polynomial algorithms are reasonable**
- **Polynomial problems are tractable**

- **Exponential algorithms are unreasonable**
- **Exponential problems are intractable**

Terminology

	Problems	*Algorithms*
Polynomial	*Tractable*	*Reasonable*
Exponential	*Intractable*	*Unreasonable*

What is an efficient algorithm?

Is an O(n) algorithm efficient?

How about O(n log n)?

$O(n^2)$?

$O(n^{10})$?

$O(n^{\log n})$?

$O(2^n)$?

O(n!) ?

polynomial time

$O(n^c)$ for some constant c

non-polynomial time

Decision Problems

To keep things simple, we will mainly concern ourselves with decision problems. These problems only require a single bit output ``yes'' and ``no''.

How would you solve the following decision problems?

Is this directed graph acyclic?

Is there a spanning tree of this undirected graph with total weight less than w?

Does this bipartite graph have a perfect (all nodes matched) matching?

Does the pattern p appear as a substring in text t?

P

P is the set of decision problems that can be solved in worst-case polynomial time:

If the input is of size n, the running time must be O(n^k).
Note that k can depend on the problem class, but not the particular instance.

All the decision problems mentioned above are in P.

Nice Puzzle

The class NP (meaning non-deterministic polynomial time) is the set of problems that might appear in a puzzle magazine: ``Nice puzzle.''

What makes these problems special is that they might be hard to solve, but a short answer can always be printed in the back, and it is easy to see that the answer is correct once you see it.

Example... Does matrix A have an LU decomposition?

No guarantee if answer is ``no''.

NP

Technically speaking:

A problem is in NP if it has a short accepting certificate:
An accepting certificate is something that we can use to quickly show that the answer is ``yes'' (if it is yes).
Quickly means in polynomial time.
Short means polynomial size.

This means that all problems in P are in NP (since we don't even need a certificate to quickly show the answer is ``yes'').

But other problems in NP may not be in P. Given an integer x, is it composite? How do we know this is in NP?

Certificates

- Returning true: in order to show that the schedule can be made, we only have to show one schedule that works
  - This is called a certificate.
- Returning false: in order to show that the schedule cannot be made, we must test all schedules.

NP-Complete

"NP-Complete" comes from:

- Nondeterministic Polynomial
- Complete - "Solve one, Solve them all"

There are more NP-Complete problems than provably intractable problems.

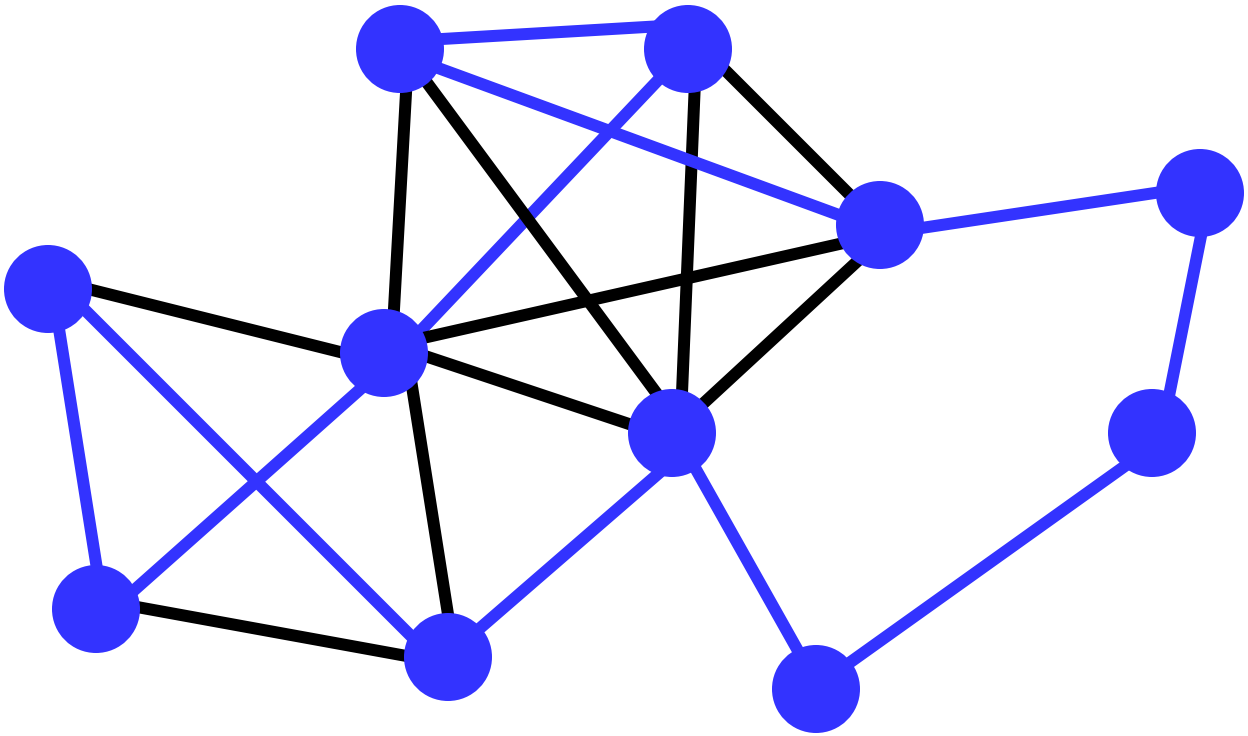
Proving NP-Completeness

- Show that the problem is in NP. (i.e. Show that a certificate can be verified in polynomial time.)
- Assume it is not NP complete
- Show how to convert an existing NPC problem into the problem that we are trying to show is NP Complete (in polynomial time).
- If we can do it we've done the proof!
- Why?
- If we can turn an existing NP-complete problem into our problem in polynomial time... $\rightarrow | \leftarrow$

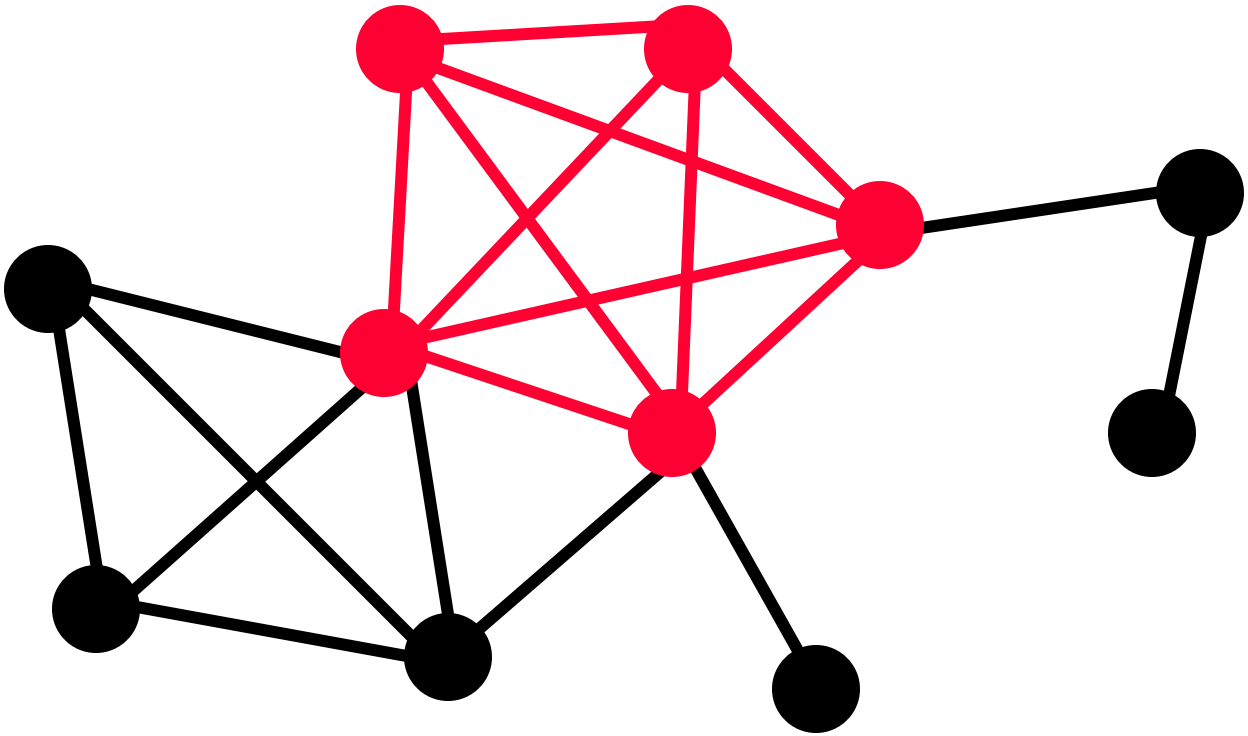
Example NP-Complete Problems

- Path-Finding (Traveling salesman)
- Scheduling
- Planning problems (pert planning)
- Clique

Traveling Salesman



5-Clique



Class Scheduling Problem

- With N teachers with certain hour restrictions M classes to be scheduled, can we:
  - Schedule all the classes
  - Make sure that no two teachers teach the same class at the same time
  - No teacher is scheduled to teach two classes at once

Some NP-complete problems

- Many practical problems are NP-complete.
  - Given a linear program (a set of linear inequalities) is there an integer solution to the variables?
  - Given a set of integers, can they be divided into two sets whose sum is equal?
  - Given two identical processors, a set of tasks of varying length, and a deadline, can the tasks be scheduled so that they finish before the deadline?
  - If there is an efficient solution to any of these, then all NP problems have efficient solutions! This would have a major impact.

P=NP or P≠NP?

- Proving whether P=NP or P≠NP is one of the most important open problems in computer science.
- If someone showed that P=NP, then many "hard" problems (i.e. The NP-complete problems) would be tractable.
- However most computer scientists believe that P≠NP, largely because there are many problems which are in NP but for which no one has found an efficient solution.
  - That is, absence of evidence that P=NP counts as evidence that P≠NP.

NP-Complete Problems

- We will see that NP-Complete problems are the "hardest" problems in NP:
  - If any *one* NP-Complete problem can be solved in polynomial time…
  - …then *every* NP-Complete problem can be solved in polynomial time…
  - …and in fact *every* problem in **NP** can be solved in polynomial time (which would show **P = NP**)
  - Thus: solve hamiltonian-cycle in $O(n^{100})$ time, you've proved that **P = NP**. Retire rich & famous.

Reduction

- The crux of NP-Completeness is *reducibility*
  - Informally, a problem P can be reduced to another problem Q if *any* instance of P can be "easily rephrased" as an instance of Q, the solution to which provides a solution to the instance of P
    - This rephrasing is called *transformation*
  - Intuitively: If P reduces to Q, P is "no harder to solve" than Q

Reducibility

- An example:
  - P: Given a set of Booleans, is at least one TRUE?
  - Q: Given a set of integers, is their sum positive?
  - Transformation: $(x_1, x_2, \ldots, x_n) = (y_1, y_2, \ldots, y_n)$ where $y_i = 1$ if x_i = TRUE, $y_i = 0$ if x_i = FALSE
- Another example:
  - Solving linear equations is reducible to solving quadratic equations

POLY-TIME *REDUCIBILITY*

A language A is polynomial time reducible to language B, written $A \leq_P B$, if there is a polynomial time computable function

$f : \Sigma^* \rightarrow \Sigma^*$, where for every w,

$$w \in A \Leftrightarrow f(w) \in B$$

f is called a polynomial time reduction of A to B

NP-complete:

- Informally, the hardest problem in class NP
  - if an NP-complete problem can be solved in polynomial time, then every problem in NP can be too
- Formally,
  - a language L_1 is polynomial-time reducible to a language L_2, if there exists a polynomial-time computable function $f : \{0,1\}^* \to \{0,1\}^*$ such that for all $x \in \{0,1\}^*$,

$$x \in L_1 \text{ if and only if } f(x) \in L_2.$$

written as $L_1 \leq_p L_2$

- Question: How can we prove a problem to be NP-complete?

  Proof by definition?

  Yes, for the first NP-complete problem, we have to.

- After we know some NP-complete problems, we can prove a new problem L to be NP-complete through polynomial-time reduction:

  1. show $L \in$ NP
  2. find an NP-complete problem L'
  3. show $L' \leq_p L$

  We conclude that L is NP-complete.

Using Reductions

- If P is *polynomial-time reducible* to Q, we denote this $P \leq_p Q$
- Definition of NP-Complete:
  - If P is NP-Complete, $P \in \mathbf{NP}$ and all problems R are reducible to P
  - Formally: $R \leq_p P \ \forall \ R \in \mathbf{NP}$
- If $P \leq_p Q$ and P is NP-Complete, Q is also NP-Complete
  - This is the *key idea* you should take away today

Other NP-Complete Problems

- Travelling Salesman Problem
- Hamiltonian Path
- Max Cut
- Subset Sum
- Integer Programming
- ….

NP-Hard and NP-Complete

- If P is *polynomial-time reducible* to Q, we denote this $P \leq_p Q$
- Definition of NP-Hard and NP-Complete:
  - If all problems $R \in$ **NP** are reducible to P, then P is *NP-Hard*
  - We say P is *NP-Complete* if P is NP-Hard and $P \in$ **NP**
- If $P \leq_p Q$ and P is NP-Complete, Q is also NP- Complete

Why Prove NP-Completeness?

- Though nobody has proven that **P** != **NP**, if you prove a problem NP-Complete, most people accept that it is probably intractable
- Therefore it can be important to prove that a problem is NP-Complete
  - Don't need to come up with an efficient algorithm
  - Can instead work on *approximation algorithms*

Proving NP-Completeness

- *What steps do we have to take to prove a problem P is NP-Complete?*
  - Pick a known NP-Complete problem Q
  - Reduce Q to P
    - Describe a transformation that maps instances of Q to instances of P, s.t. “yes” for P = “yes” for Q
    - Prove the transformation works
    - Prove it runs in polynomial time

6. ***$\mathcal{NP}$-Completeness Basics*** **[12 points]**

(a) Please indicate which classes these problems belong in.

Problem	$\mathcal{P}$	$\mathcal{NP}$	$\mathcal{NP}$-Complete
SUBSET SUM	NO	YES	YES
STRING MATCHING	YES	YES	NO

Assume $\mathcal{P} \neq \mathcal{NP}$.

(b)] Let A and B be decision problems such that $A \leq_p B$. Answer true or false.

i. If $B \in \mathcal{P}$ then $A \in \mathcal{P}$. TRUE

ii. If B is $\mathcal{NP}$-complete then A is $\mathcal{NP}$-complete. FALSE

iii. If B can be decided in $O(n^3)$ time then A can be decided in $O(n^3)$ time.

FALSE

Given an undirected graph $G = (V, E)$, a set of vertices $W \subseteq V$ is a *clique* if for all $u, v \in W, (u, v) \in E$. *In other words, there is an edge between every pair of vertices in* W.

Given an undirected graph $G = (V, E)$, a set of vertices $W \subseteq V$ is an *independent set* if for all $u, v \in W, (u, v) \notin E$. *In other words, there is no edge between any pair of vertices in* W.

Consider the CLIQUE and INDEPENDENT SET (IS) problems for undirected graphs that we studied in class.

CLIQUE $= \{(G, k) \mid G$ has a clique of size $k\}$

IS $= \{(G, k) \mid G$ has an independent set of size $k\}$

We now define the new problem ISCLIQUE.

ISCLIQUE $= \{(G, k) \mid G$ has a clique of size k *and* an independent set of size $k\}$.

(a) Define a certificate for ISCLIQUE. Show that we can *verify* the certificate in deterministic polynomial time.

A certificate for ISCLIQUE is

2 sets $W_1, W_2 \subseteq V(G)$.

① for all $u, v \in W_1$, check if $(u, v) \in E(G)$. $O(n^2)$

② for all $u, v \in W_2$, check if $(u, v) \notin E(G)$. $O(n^2)$

③ check if $|W_1| = k$ $O(n)$

④ check if $|W_2| = k$ $O(n)$

(b) Consider an undirected graph G and an integer k. Construct a new graph H from G by adding k vertices to G but no additional edges. So, $G = (V, E)$ and $H = (V \cup W, E)$ where W is a set of k new vertices.

i. Show that if $(G, k) \in$ CLIQUE then $(H, k) \in$ ISCLIQUE.

Suppose $(G, k) \in$ CLIQUE. Let $C \subseteq V$ be the clique of size k in G. Since $C \subseteq V(H)$, C is also a clique in H. Also, W is an independent set of size k in H. So, H has a clique of size k and an independent set of size k, implying that $(H, k) \in$ ISCLIQUE.

ii. Show that if $(H, k) \in$ ISCLIQUE then $(G, k) \in$ CLIQUE.

Suppose $(H, k) \in$ ISCLIQUE. Let $C \subseteq V \cup W$ be a clique of size k in H. Since there are no edges incident on any vertex in W, no vertex in W is in C. So, $C \subseteq V$ and C is a clique of size k in G, implying that $(G, k) \in$ CLIQUE.

(c) **[6 pts]** What have we shown in part (a) and in part (b)? Using the fact that CLIQUE is NP-complete, what can we now conclude?

(a) $\Rightarrow$ ISCLIQUE $\in$ NP

(b) $\Rightarrow$ CLIQUE $\leq_p$ ISCLIQUE

$\Rightarrow$ ISCLIQUE is NP-complete